Non Maskable Interrupt

The NMI ("Non Maskable Interrupt") is a hardware-driven interrupt much like the PIC interrupts, but the NMI goes either directly to the CPU, or via another controller (e.g., the ISP)---in which case you can mask them.

About

NMI occur for RAM errors and unrecoverable hardware problems. For newer computers these things may be handled using machine check exceptions and/or SMI. For the newest chipsets (at least for Intel) there's also a pile of TCO stuff ("total cost of ownership") that is tied into it all (with a special "TCO IRQ" and connections to SMI/SMM, etc). Somehow all of the TCO stuff is/can be connected to an onboard ethernet controller, and (at least part of it) is intended for remote monitoring of the system. Unfortunately the chipset documentation I've been reading can't tell me how BIOSs normally configure the chipset, and the chipsets themselves support several different options in each case. For example, for a RAM error it could be handled by the chipset itself, it could generate an SMI (where the BIOS/SMM handler does "RAM scrubbing" in software), it could generate a "TCO interrupt", etc. If you add it all up it's a huge complex mess (TCO + SMI + SMBus + northbridge + PCI bus/controller/s + PCI-to-LPC-bridge + god-knows-what) that can be completely different between motherboards (even motherboards with the same chipset).

The short version of this story is that there's only really 2 reasons for an NMI. The first reason is a hardware failure. The second reason is a "watchdog timer", which can be used to detect when the kernel itself locks up (and is sometimes also used for more accurate profiling as it allows EIP to be sampled even when IRQs are disabled).

If a hardware failure caused an NMI then there's no way to figure out which piece of hardware caused the NMI. In this case I'd try to do the least possible in an attempt to tell the user that a hardware failure occurred, but at the end of the day you can't expect any OS to work sanely on faulty hardware and there's nothing software can do to work around the hardware failure anyway.

For the watchdog timer, it must be setup by the OS first. This can actually be done even when the chipset itself doesn't have a special watchdog timer for it (e.g. setting the PIT, RTC/CMOS IRQ or a HPET IRQ to "NMI, send to all CPUs" in the I/O APIC). In this case you want the watchdog timer to be fast (i.e. no slow hardware task switching and cache flushing) and you'd also want all CPUs to share the same timer, which means all CPUs would receive the same IRQ at the same time (which brings me back to the busy flag in your TSS).

As an alternative, you could also use the local APIC's timer or the performance monitoring counter overflow for a "per CPU" watchdog timer. Unfortunately these things are usually used for other purposes.

Usage

The NMI is "turned on" (set high) by the memory module when a memory parity error occurs.

You have to be careful about disabling the NMI and the PIC for extended periods of time (mind you, watchdog timers typically use NMIs).

```c
void NMI_enable(void)
{
   outb(0x70, inb(0x70)&0x7F);
}

void NMI_disable(void)
{
   outb(0x70, inb(0x70)|0x80);
}
```

When an NMI occurs you can check the system control port A and B at io addresses 0x92 and 0x61 respectively to get an indication of what caused the error:

System Control Port A (0x92) layout:

BIT	Description
0	Alternate hot reset
1	Alternate gate A20
2	Reserved
3	Security Lock
4*	Watchdog timer status
5	Reserved
6	HDD 2 drive activity
7	HDD 1 drive activity

System Control Port B (0x61)

Bit	Description
0	Timer 2 tied to speaker
1	Speaker data enable
2	Parity check enable
3	Channel check enable
4	Refresh request
5	Timer 2 output
6*	Channel check
7*	Parity check

The important bits are indicated with an '*'. The Channel Check bit indicates a failure on the bus, probably by a peripheral device such as a modem, sound card, NIC, etc, while the Parity check bit indicates a memory read or write failure.